Noname manuscript No.
(will be inserted by the editor)

Hardening against adversarial examples with the Smooth Gradient Method

Alan Mosca · George D. Magoulas

Received: date / Accepted: date

Abstract Commonly used methods in Deep Learning do not utilise transformations of the residual gradient available at the inputs to update the representation in the dataset. It has been shown that this residual gradient, which can be interpreted as the first order gradient of the input sensitivity at a particular point, may be used to improve generalisation in feed-forward neural networks, including fully connected and convolutional layers. We explore how these input gradients are related to input perturbations used to generate *adversarial examples*, and how the networks that are trained with this technique are more robust to attacks generated with the Fast Gradient Sign method.

1 Introduction

Most supervised Deep Learning models are trained with backpropagation [22, 6], and different variants of Gradient Descent. The main principle upon which this methodology is based is the application of a so-called *weights update rule*, which uses an error gradient with respect to the internal parameters of the model

$$\Delta \boldsymbol{W_t} = \nabla E\left(\boldsymbol{W_t}\right) \tag{1}$$

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GTX Titan X GPUs used for this research.

Alan Mosca
Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, London WC1E 7HX, E-mail: a.mosca@dcs.bbk.ac.uk

George D. Magoulas
Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, London WC1E 7HX, E-mail: gmagoulas@dcs.bbk.ac.uk

at training epoch t, where $\boldsymbol{W_t}$ are the parameters of the hypothesis $h(\boldsymbol{W_t}, X)$ being optimised, where $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ is the input matrix. Such an update rule $\boldsymbol{\Delta W_t}$ is used to produce an incremental parameter change:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \lambda \nabla E_t \left(\boldsymbol{W_t}\right) \tag{2}$$

where λ is usually called the learning rate. The function E_t generates the values of the *error function* at time t, sometimes also referred to as the *cost function*.

For simplicity of notation, we indicate nodes with a single index, rather than identifying them by layer and position in the layer. In practice, when considering Artificial Neural Networks trained with back-propagation, each parameter $w_{ki \to lj,t}$, which represents the weight connecting the i-th node of the k-th layer to the j-th node of the $l = (k+1)$-th layer, $\forall l \in 1, ...N$ is updated by considering the corresponding element of the Jacobian i.e the partial derivative

$$\frac{\partial E_t}{\partial w_{ki \to lj,t}} \tag{3}$$

of the error function E_t with respect to each weight. This information is used in the calculation of $w_{ki \to lj,t+1}$.

When back-propagated, the calculation of the partial derivative of Equation (3) is based on application of the chain rule, as shown in Equation (4) below, and typically stops at the first/input layer $k = 0$. N is the number of layers in the network.

$$\frac{\partial E_t}{\partial w_{0i \to 1j,t}} = \frac{\partial w_{1i \to 2j,t}}{\partial w_{0i \to 1j,t}} \cdots \frac{\partial w_{(N-1)i \to Nj,t}}{\partial w_{(N-2)i \to (N-2)j,t}} \frac{\partial E_t}{\partial w_{(N-1)i \to Nj,t}} \tag{4}$$

The term $\frac{\partial w_{1i \to 2j,t}}{\partial w_{0i \to 1j,t}}$ of the equation is some times called *input sensitivity*. If we abbreviate this term as S_t, then we can reformulate Equation (4) in terms of the input sensitivity, as shown in Equation (5):

$$\frac{\partial E_t}{\partial w_{0i \to 1j,t}} = S_t \frac{\partial w_{2i \to 3j,t}}{\partial w_{1i \to 2j,t}} \cdots \frac{\partial w_{(N-1)i \to Nj,t}}{\partial w_{(N-2)i \to (N-1)j,t}} \frac{\partial E_t}{\partial w_{(N-1)i \to Nj,t}} = S_t \frac{\partial E_t}{\partial w_{1i \to 2j,t}} \tag{5}$$

1.1 Deep Neural Networks

Deep Neural Networks (DNN) follow basic principles of traditional Artificial Neural Networks but include a large number of layers. An example of such a methodology is shown in Ref [2].

The backpropagation profile of these networks works as follows. Given a set of weights $w_{ki \to lj}$, a node value net_j and a continuous differentiable activation

function $\varphi(x)$ [1], then the output $o_{lj,t}$ of a node j in a hidden layer l at time t of a DNN is

$$o_{lj,t} = \varphi(\text{net}_{lj,t}) = \varphi\left(\sum_{i=1}^{n} w_{ki\to lj,t} o_i\right) \tag{6}$$

and the derivative of the error with respect to the weights of a node in the output layer can be rewritten as

$$\frac{\partial E_t}{\partial w_{ki\to lj,t}} = \frac{\partial E_t}{\partial o_{lj,t}} \frac{\partial o_{lj,t}}{\partial \text{net}_{lj,t}} \frac{\partial \text{net}_{lj,t}}{\partial w_{ki\to lj,t}}. \tag{7}$$

For all n nodes in the previous layer $i = l - 1$, the last term becomes

$$\frac{\partial \text{net}_{lj,t}}{\partial w_{ki\to lj,t}} = \frac{\partial}{\partial w_{ki\to lj,t}} \left(\sum_{z=1}^{n} w_{kz\to lj,t} o_{kz,t}\right) = o_{ki,t}. \tag{8}$$

The second term becomes

$$\frac{\partial o_{lj,t}}{\partial \text{net}_{lj,t}} = \frac{\partial}{\partial \text{net}_{lj,t}} \varphi(\text{net}_{lj,t}) \tag{9}$$

and for each node p in a layer q, the first term becomes

$$\frac{\partial E_t}{\partial o_{lj,t}} = \sum_{q=1\to l-1} \sum_{p\in q} \left(\frac{\partial E_t}{\partial \text{net}_{qp,t}} \frac{\partial \text{net}_{qp,t}}{\partial o_{lj,t}}\right) = \sum_{q=1\to l-1} \sum_{p\in q} \left(\frac{\partial E_t}{\partial o_{qp,t}} \frac{\partial o_{qp,t}}{\partial \text{net}_{qp,t}} w_{mp\to lj,t}\right), \tag{10}$$

1.2 Convolutional Neural Networks

Convolutional Neural Networks [12] are normally composed of two parts: convolution and pooling. Convolution creates a sliding window over the input space which is convolved to learn a shared set of weights that produces a smaller output. Pooling reduces the number of features by selecting only the maximum out of a pre-defined *pool* of the outputs of the convolutional step. For convenience we consider convolution in two dimensions, however the procedure can be extended to an arbitrary number of dimensions.

If we consider an $N \times M$ rectangular output from layer node i in layer k to node j in layer l, to which a $n \times m$ rectangular kernel $W_{ki\to lj,t}$ is applied, the forward pass of the convolution operation to produce an output value $o_{l\boldsymbol{j},t}$, after applying an activation function $\varphi(\cdot)$, yields:

$$o_{l\boldsymbol{j},t} = \varphi\left(\sum_{a=0}^{n-1} \sum_{b=0}^{m-1} w_{ab} o_{k(\boldsymbol{j}+\{a,b\})}\right), \tag{11}$$

[1] in some cases activation functions with a single non-differentiable point with no discontinuity has been shown to still work [5]

where $w_{ki \to lj,ab,t}$ is the weight contained in the kernel W at coordinates a, b, and $\boldsymbol{j} = \{j_0, j_1\}$ are the coordinates of the output value. This makes the gradient more complex, but still computable:

$$\begin{aligned}\frac{\partial E_t}{\partial w_{ki \to lj,ab,t}} &= \sum_{a=0}^{n-1}\sum_{b=0}^{m-1} \frac{\partial E_t}{\partial o_{l\boldsymbol{j},t}} \frac{\partial o_{l\boldsymbol{j},t}}{\partial w_{ki \to lj,ab,t}} \\ &= \sum_{a=0}^{n-1}\sum_{b=0}^{m-1} \frac{\partial E_t}{\partial o_{l\boldsymbol{j},t}} \varphi'\left(o_{k(\boldsymbol{j}+\{a,b\},t}\right) \varphi\left(o_{k(\boldsymbol{j}+\{a,b\},t}\right)\end{aligned} \tag{12}$$

this can be propagated backwards, obtaining:

$$\begin{aligned}\frac{\partial E_t}{\partial o_{k\boldsymbol{j},t}} &= \sum_{a=0}^{n-1}\sum_{b=0}^{m-1} \frac{\partial E_t}{\partial o_{l(\boldsymbol{j}-\{a,b\}),t}} \frac{\partial o_{l(\boldsymbol{j}-\{a,b\}),t}}{\partial o_{k\boldsymbol{j},t}} \\ &= \sum_{a=0}^{n-1}\sum_{b=0}^{m-1} \frac{\partial E_t}{\partial o_{l(\boldsymbol{j}-\{a,b\}),t}} w_{ki \to lj,ab,t}\end{aligned} \tag{13}$$

The rest of the paper is structured as follows. Section 2 explores the theory behind the technique and introduces new considerations in merit. Section 3 explores the relationship between learning input representations and adversarial examples. Section 4 extends the theory to random perturbations of the input (such as random noise). Section 5 shows experimental results on benchmark datasets in computer vision. Section 6 shows experimentally how the networks generated develop a higher robustness to adversarial examples, and Section 7 draws final conclusions about the technique and the associated experimental results.

2 Backpropagating the error to the input

By the same principle, the chain rule can be applied once more, obtaining a new partial derivative of the error with respect to the specific representation of each component of the input matrix $\boldsymbol{X}$ at epoch t, as shown in Equation (14). We call this the *residual error gradient at the inputs*.

$$\frac{\partial E_t}{\partial x_{i,t}} = \frac{\partial E_t}{\partial w_{0i \to 1j,t}} \frac{\partial w_{0i \to 1j,t}}{\partial x_{i,t}}. \tag{14}$$

where $w_{0i \to 1j,t}$ is the weight that connects input i to node j in the first layer.

The idea of forming global representations of the input patterns has been previously investigated in the FGREP model [14], where an external lexicon network is continuously updated by extending the formulation of Backpropagation to the input representations stored in the lexicon. This work lays the foundation upon which we can construct a more general formulation that can be extended to adapt the representation of the training inputs of the Artificial Neural Network, rather than a randomly initialised lexicon.

This approach has been further developed in Ref [17], where the principle of applying one further step of backpropagation to obtain an update of the inputs in convolutional and fully-connected deep neural networks has been introduced. The authors also provide a proof of computability of the residual error of Equation (14). An important caveat is that, because $\frac{\partial E_t}{\partial x_{i,t}}$ is determined analytically, the proof does not hold when random perturbations of the input are present, such as random noise injection or elastic transformations [23]. We will however show that, when the random perturbations are designed to have a mean converging towards zero, this approach can still be applied, and as our experiments confirm, it is able to still provide comparable generalisation capability.

The residual error of Equation (14) can be utilised to update the value of the inputs at each training epoch to increase the speed of convergence, with the goal of maximising the internal gradients of the network at the following epoch $t + 1$. Furthermore, because we are operating on the very first inputs into the network, we do not need to distinguish between different types of networks.

Therefore an iterative scheme to adapt the input representations at each time step t can be expressed as

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \bar{\lambda} \nabla E_t \left(\boldsymbol{X}_t \right), \tag{15}$$

where $\bar{\lambda}$ is a chosen *input update learning rate*. While previous work [17] recommended that $0 \leq \bar{\lambda} \leq 1$ and a value of $\bar{\lambda} = 1$ worked well in our experiments, after further experimentation we found that values in the range $-1 \leq \bar{\lambda} \leq 1$ also work well and that the best available value of $\bar{\lambda}$ can be searched by hyperparameter optimisation.

3 Relationship to adversarial examples

Adversarial Examples [25] are a particular type of learned input perturbation that, although being imperceptible to the human eye, and therefore very small in magnitude when compared with the undisturbed input values, are able to modify the output of a neural network significantly. In some cases these examples have been likened to the neural network's equivalent of an optical illusion. In Ref [4] the authors attempt to explain this phenomenon in terms of the linear nature of the neural networks, and provide an example of how such examples can be generated. In Ref [19] the authors illustrate how such examples could become dangerous, especially as deep learning and neural networks become more prominent in the modern technological landscape.

Partial solutions exist, in the form of *adversarial training* (the act of generating several adversarial examples from regular training examples and adding them to each training epoch), and *defensive distillation* [20], where a vulnerable model is run through the process of distillation [7] to reduce the sensitivity to adversarial examples.

The generation of such adversarial examples relies on the attacker having access to the gradients of the model. However, several approaches aimed at hiding this gradient, sometimes called "gradient masking", have been shown to not be effective at reducing the sensitivity to these examples [19].

Given that the approach towards generating and protecting from adversarial examples is very similar to that of adapting the input representations, as explained in Ref [17], we analyse the relationship between the two approaches.

In both cases, there is an original traning example $\boldsymbol{x}_m$, taken from a training set $\boldsymbol{X}$, which is then combined with a perturbation $\boldsymbol{v}_m$, which is static with respect to the final network for generating an adversarial example $\tilde{\boldsymbol{x}}_m$, and pertains to the current training epoch in the case of input updates $\boldsymbol{x}_{i,t}$, taking the form of $\boldsymbol{v}_{i,t}$. Equation (16) shows the approach for adversarial examples, whilst Equation (17) shows the related expression for the training input updates.

$$\tilde{\boldsymbol{x}}_m = \boldsymbol{x}_m + \boldsymbol{v}_m \tag{16}$$

$$\boldsymbol{x}_{i,t+1} = \boldsymbol{x}_{i,t} + \boldsymbol{v}_{i,t} \tag{17}$$

Because of the similarity between Equation (16) and Equation (17), a more general notation that applies to both cases can be used to derive the expression in Equation (18), for the entire training set, with the special case that for updated inputs at each epoch, $\boldsymbol{X}_{t+1} = \tilde{\boldsymbol{X}}_t$.

$$\tilde{\boldsymbol{X}}_t = \boldsymbol{X}_t + \boldsymbol{\Upsilon}_t \tag{18}$$

In both cases, we know that Υ is derived from the gradients of the model, defined as sensitivity of the model with respect to the input changes, as shown in Equation 19, where $F(\cdot)$ is some arbitrarily chosen function.

$$\boldsymbol{\Upsilon} = F\left(\nabla E\left(\boldsymbol{X}_t\right)\right) \tag{19}$$

3.1 Current approaches

In the so-called "Fast Gradient Sign method" [4] for generating the adversarial examples, the sign of the partial derivatives of the cost with respect to the input is applied as an input perturbation, as shown in Equation (20). It is to be noted that the notation used by the authors is different from that used in this paper, and a translated version to the notation used in this paper is provided instead.

$$\tilde{\boldsymbol{x}}_{i,t} = \boldsymbol{x}_{i,t} + \bar{\lambda}\,\mathrm{sgn}\left(\frac{\partial E_t}{\partial \boldsymbol{x}_{i,t}}\right) \tag{20}$$

In the Domain-adversarial Neural Networks [3], the authors take a *domain-based* approach to determine a new type of network that utilises a "gradient

reversal layer" to ensure that feature distributions over the source and target domains are made similar. This type of approach, whilst not explicitly targeted at adversarial examples, illustrates that other solutions may exist that are not based on modified input values.

3.2 A smooth approach

By following the method prescribed in Ref [17], we are adding the partial derivative with respect to the input to the input itself at each epoch, multiplied by a learning rate $\bar{\lambda}$, Equation (21), therefore forcing the network to reduce its sensitivity to these variations, by forcing larger updates during the following round.

$$\boldsymbol{x}_{i,t+1} = \boldsymbol{x}_{i,t} + \bar{\lambda}\frac{\partial E_t}{\partial \boldsymbol{x}_{i,t}} \tag{21}$$

If we discard the application of the $\mathrm{sgn}(\cdot)$ function, the relation between the two operations is self-evident. Training the network on inputs that are continuously updated to reduce the sensitivity to the gradient of the error with respect to the input can therefore be considered equivalent to the continuous online creation of new adversarial examples at each training iterarion, against which the network is hardened. This approach, which we call the Smooth Gradient Method (SGM) leads to more robust networks and in some cases it is also able to increase traning speed and accuracy.

By not applying the $\mathrm{sgn}(\cdot)$, the input perturbations are no longer limited to the domain $\{-\bar{\lambda}, \bar{\lambda}\}$, but are instead able to extend to the entire domain of $\mathbb{R}$.

4 Extension to random perturbations

For a learning setting with random noise perturbations, we must consider the asymptotic effect of the additional factor $\boldsymbol{\epsilon}_t$ and how it will impact the considerations made for the improvements in training. It has been shown that under certain conditions the addition of particular types of training noise to the training data is very useful to improve the training performace [23]. Previous work has been done to prove the convergence on perturbations in back-propagation, and the goal of this section is to illustrate how the concept applies to the perturbation $\boldsymbol{\epsilon}_t$. We assume that $\boldsymbol{\epsilon}_t$ is a random, stationary ergodic process (in most cases of perturbations this actually the case). This will make the sequence of $\boldsymbol{X}_0 ... \boldsymbol{X}_t$ a stationary process.

If one extends Equation (18) to include a perturbation from random affectations of the training set $\boldsymbol{\epsilon}_t$, caused by a noisy learning process at epoch t, one can obtain the formulation of Equation (22).

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{\Upsilon}_t + \boldsymbol{\epsilon}_t \tag{22}$$

A brief explanation of why both the FGSM method and SGM work in a noisy setting is provided.

We begin by considering the simpler case for $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{\epsilon}_t$. At epoch t, we expect that the gradient of the error with respect to the current input will be, in expectation, similar to the error with respect to the original input $\boldsymbol{X}_0 = \boldsymbol{X}$.

For an input $x_{i,t} \in \boldsymbol{X}_t$, we assume our distribution of each perturbation $\epsilon_{i,t} \in \boldsymbol{\epsilon}_t$ to also have zero-mean and be symmetric such that

$$\mathbb{E}\left[\boldsymbol{\epsilon}_t\right] = 0 \tag{23}$$

and the variance to be such that its sum over t converges to a finite number.

$$\sum_{k=0}^{t} \mathrm{Var}\left(\boldsymbol{\epsilon}_t\right) = s \tag{24}$$

Most random noise is of this kind, as it is usually generated from samples from a gaussian distribution. We also know that

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{\epsilon}_t \tag{25}$$

If we then unroll the recursive relationship in Equation (25), we obtain

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_0 + \boldsymbol{\epsilon}_0 + \boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_{t-1} + \boldsymbol{\epsilon}_t \tag{26}$$

or, more simply

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\epsilon}_k \tag{27}$$

We can then rewrite our problem statement as

$$\mathbb{E}\left[\sum_{k=0}^{t} \frac{\partial E_k}{\partial \boldsymbol{X}_t}\right] = \mathbb{E}\left[\sum_{k=0}^{t} \frac{\partial E_k}{\partial \left(\boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\epsilon}_k\right)}\right] = \mathbb{E}\left[\sum_{k=0}^{t} \frac{\partial E_k}{\partial \left(\boldsymbol{X}_0 + \mathbb{E}\left[\sum_{k=0}^{t} \boldsymbol{\epsilon}_k\right]\right)}\right] \tag{28}$$

By the definition of its distribution, we know that

$$\mathbb{E}\left[\sum_{k=0}^{t} \boldsymbol{\epsilon}_k\right] = \sum_{k=0}^{t} \mathbb{E}\left[\boldsymbol{\epsilon}_k\right] = 0 \tag{29}$$

and that this is admissible because we restricted the sequence of all $\boldsymbol{\epsilon}_t$ to have a cumulative converging variance (Equation 24, therefore allowing that

$$\mathbb{E}\left[\sum_{k=0}^{t} \frac{\partial E_k}{\partial \boldsymbol{X}_t}\right] \approx \mathbb{E}\left[\sum_{k=0}^{t} \frac{\partial E_k}{\partial \boldsymbol{X}_0}\right] \tag{30}$$

We now consider the case for $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \boldsymbol{\Upsilon}_t + \boldsymbol{\epsilon}_t$, which can be stated as

$$\mathbb{E}\left[\nabla E_t\left(\boldsymbol{X}_t\right)\right] \approx \mathbb{E}\left[\nabla E_t\left(\boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\Upsilon}_k\right)\right] \tag{31}$$

and simplified to

$$\mathbb{E}\left[\boldsymbol{X}_t\right] \approx \mathbb{E}\left[\boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\Upsilon}_k\right] \tag{32}$$

From a similar unrolling procedure to Equation (26), we can derive that

$$\boldsymbol{X}_t = \boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\Upsilon}_k + \sum_{k=0}^{t} \boldsymbol{\epsilon}_k \tag{33}$$

and by Equation (29)

$$\mathbb{E}\left[\boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\Upsilon}_k + \sum_{k=0}^{t} \boldsymbol{\epsilon}_k\right] = \mathbb{E}\left[\boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\Upsilon}_k\right] \tag{34}$$

which in turn allows for

$$\mathbb{E}\left[\boldsymbol{X}_t\right] = \mathbb{E}\left[\boldsymbol{X}_0 + \sum_{k=0}^{t} \boldsymbol{\Upsilon}_k\right] \tag{35}$$

Therefore the original statement in Equation (32) can be considered valid.

5 Experimental analysis

The input update technique has been tested on some common benchmark computer vision datasets, frequently used to compare the performance of Deep Learning algorithms. Using computer vision tasks allows us also to visualise the gradients propagated back to the input, the updated inputs and the adversarial examples in a human–friendly manner. Our experimental methodology consists of running the same Convolutional Neural Network five times and reporting the median results. This is the same methodology as reported in [24]. We chose this methodology because it is in line with other results on the same datasets, and it allows us to produce effective results even in the case where the training is a very slow process [2]. We first generate our control experiment without augmenting the input, and we then repeat the same experiments with the updates enabled. For consistency, the random initializations are fixed to be in lock-step: the N^{th} run of each of the both variants of the architecture would receive the same initialisation.

For all network types we used the WAME weight update rule [18], which has been shown to have good properties of fast convergence on Convolutional Neural Networks.

[2] In certain cases a single network spent upwards of 24hrs to train on a single GPU.

2500 fully-connected nodes 50% dropout
2000 fully-connected nodes 50% dropout
1500 fully-connected nodes 50% dropout
1000 fully-connected nodes 50% dropout
500 fully-connected nodes 50% dropout
10 way softmax

Table 1: The fully-connected network structure used for the MNIST dataset.

For the input update methodology, after experimental tuning of hyperparameters, an input learning rate of $\lambda = 0.2$ was used to maximise the robustness to adversarial examples.

5.1 Datasets

The datasets used for comparing our technique are common computer vision datasets, frequently used to compare performance of supervised Deep Learning methods. They are as follows.

5.1.1 MNIST

MNIST [13] is a common computer vision dataset that associates pre-processed images of hand-written numerical digits with a class label representing that digit. The input features are the raw pixel values for the 28×28 images, in grayscale, and the outputs are the numerical value between 0 and 9. There are 50000 training images, 10000 validation images and 10000 test images.

For this dataset, we tested both a Deep Neural Network built only with fully-connected layers, and a Convolutional Neural Network.

Fully-connected Deep Neural Networks: For the Fully-connected version of the MNIST network, we used the network structure shown in Table 1.

All layers used batch normalization [9].

Figure 1 shows that the mean test misclassification at each epoch closely tracks that of the control experiment, and then continues to be reduced after the control experiment has stopped improving.

Convolutional Neural Networks: For the Convolutional version of the MNIST network we used the structure illustrated in Table 2.

All layers used batch normalization.

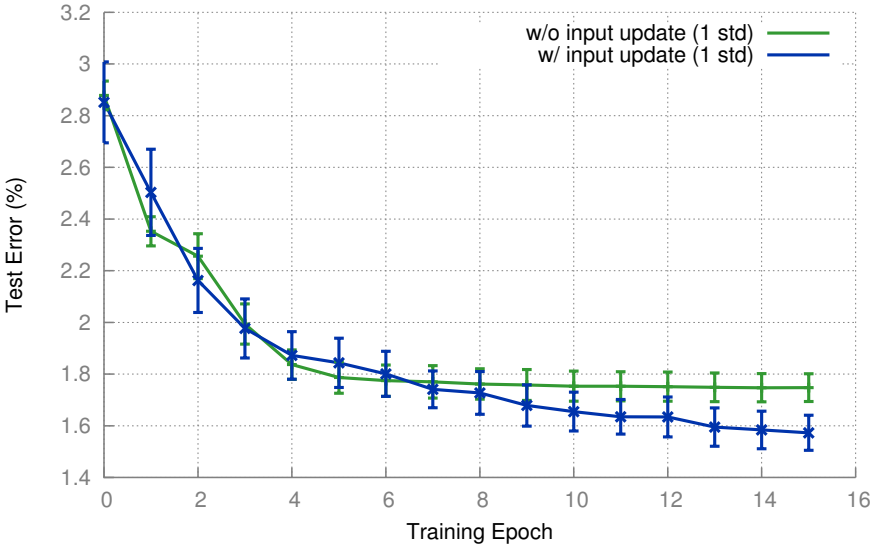

Fig. 1: Test misclassification rate - DNN on MNIST, without dataset augmentation

64 conv, 5×5
2×2 max-pooling
64 conv, 5×5
2×2 max-pooling
1024 fully-connected nodes
50% dropout
10-way softmax

Table 2: The convolutional network structure used for the MNIST dataset.

Figure 2 shows that the mean test misclassification at each epoch during the initial phase of the learning closely tracks that of the control experiment, and is in most cases marginally better.

5.1.2 CIFAR-10

CIFAR-10 is a dataset that contains 60000 small images of 10 categories of objects. It was first introduced in [11]. The images are 32×32 pixels, in RGB format. The output categories are *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*. The classes are completely mutually exclusive so that it is translatable to a *1-vs-all* multiclass classification. Of the 60000 samples, there is a training set of 40000 instances, a validation set of 10000 and a test set of another 10000. All sets have perfect class balance.

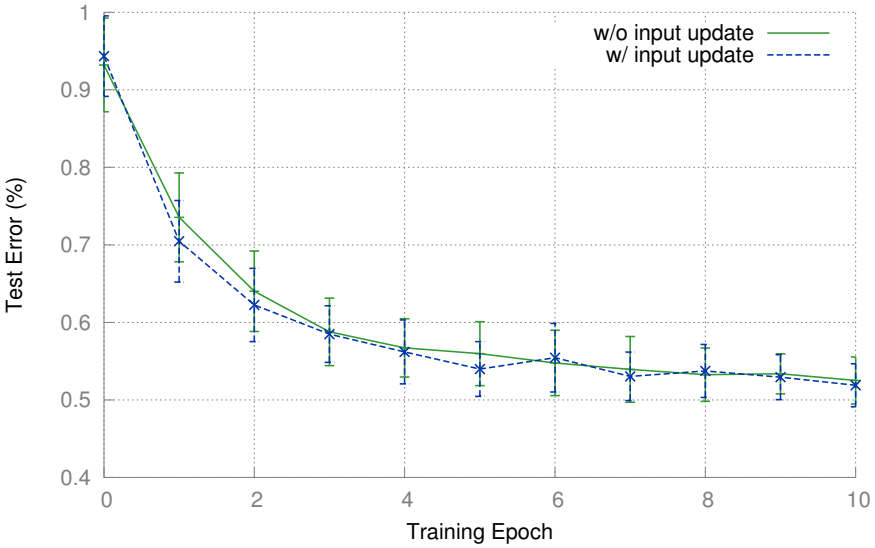

Fig. 2: Test misclassification rate - CNN on MNIST, without dataset augmentation

2×96 conv, 3×3
96 conv, 3×3, 2×2 strides
96 conv, 3×3, 2×2 strides
96 conv, 3×3, 2×2 strides
2×2 max-pooling
2×192 conv, 3×3
192 conv, 3×3, 2×2 strides
192 conv, 3×3, 2×2 strides
192 conv, 3×3, 2×2 strides
2×2 max-pooling *
192 conv, 3×3
192 conv, 1×1
10 conv, 1×1
global average pooling
10-way softmax

Table 3: The network structure used for the CIFAR-10 dataset.

The CNN we used for CIFAR-10 is a variant of the model used in Ref [15], which is a relatively small network that reaches a very good accuracy in a short training time. Its structure is illustrated in Table 3.

5.1.3 CIFAR-100

CIFAR100 is a dataset that contains 60000 small images of 100 categories of objects, grouped in 20 super-classes. It was first introduced in Ref [11]. The

	Control	With Updates
MNIST-DNN	98.81 %	98.93 %
MNIST-DNN-AUG	99.03 %	99.01 %
MNIST-CNN	99.48 %	99.49 %
MNIST-CNN-AUG	99.55 %	99.64 %
CIFAR-10-CNN	90.51 %	90.48 %
CIFAR-10-CNN-AUG	93.06 %	93.11 %
CIFAR-100-CNN	65.87 %	65.84 %
CIFAR-100-CNN-AUG	70.02 %	70.45 %

Table 4: Test accuracy on the benchmark datasets for the methods compared. DNN indicates a fully-connected Deep Neural Network, CNN indicates a Convolutional Neural Network, whilst AUG indicates that dataset augmentation was used.

image format is the same as CIFAR10. Class labels are provided for the 100 classes as well as the 20 super-classes. A super-class is a category that includes 5 of the fine-grained class labels (e.g. "insects" contains *bee, beetle, butterfly, caterpillar, cockroach*). Of the 60000 samples, there is a training set of 40000 instances, a validation set of 10000 and a test set of another 10000. All sets have perfect class balance.

The model we used was the same as that used for the CIFAR-10 dataset, illustrated in Table 3.

5.2 Data augmentation

The augmentation of available data with additional derived examples, such as from linear transformation (flip, rotate, zoom, random cropping) or more complex ones, is a common practice with all the datasets used in this paper. In the case of MNIST, the application of dataset augmentation practices is indeed very common. A specialised methodology, called *elastic distortions* [23], has been frequently used in conjunction with this dataset, to generate a virtually-infinite training dataset in an "online" fashion. At each epoch a new distorted version of the training set is created, by simulating hand jitters and other non-linear imperfections. In our results we include both augmented and non-augmented versions of the dataset, for both types of networks.

For CIFAR-10 and CIFAR-100, we utilised ZCA Whitening as a preprocessing method, and then used a light augmentation schedule of random rotations, zoom, and random horizontal flips. We did not use any random cropping.

5.3 Results

Table 4 shows accuracy on the unperturbed (non-adversarial) test set, for both our methodology and the control experiment. It is evident that our methodology does not affect the learning negatively, because most of the accuracy

values have been actually improved slightly. In addition, a Wilcoxon test rejects the hypothesis that the two experiments are significantly different with respect to this accuracy measurement. It has been shown in the literature [17] that using $\bar{\lambda} = 1$ would make such an improvement significant, and as such our preference for optimising this hyperparameter has been given to learning robustness to adversarial examples.

6 Testing robustness to adversarial examples

We constructed an adversarial test set, following the procedure explained in [4]: we added perturbations to the original test set, based on the Fast Gradient Sign method, and tested accuracy of predictions for each model based on the dataset. We then compared how each of the trained models fared on this new test set. We used different values of $\bar{\lambda} = 0.001, 0.01, 0.1, 0.2$ for comparison with small and large perturbations.

Results are compiled in Table 5. A Wilcoxon paired test shows that the improvements on adversarial examples are statistically significant at the 0.005 confidence level, giving strong reason to suggest that using the input update method is able to improve robustness to adversarial examples generated with the Fast Gradient Sign method.

The images in Figures 3 and 4, show images that have been misclassified by the original classifier, but have been correctly classified by the classifier hardened using our method. The adversarial images are still clearly visible as their original class by the human eye, whilst the classifier is making significant mistakes – for example the airplane in Figure 4 is being classified as *automobile* at both levels of $\bar{\lambda}$ being shown. An image of the perturbations alone is also included, for reference. Even though the figures have been inverted to make the distortions more evident, in many cases they are still mostly imperceptible.

Figures 5 and 6 show examples of backpropagated input distortions during the first nine epochs, for MNIST and CIFAR-10 respectively. It is of note that the updates to the input appear to be small in magnitude, but nonetheless their effect is noticeable [3].

7 Conclusions

We have shown how adapting the input representations based on the residual gradient of the error with respect to the inputs at time t, is not only an effective technique for increasing the speed of convergence and overall learning capability of an Artificial Neural Network, as was previously shown by Ref [17], but that the application of this process with a smaller input learning rate $\bar{\lambda}$

[3] 50% grey indicates no update, whilst white indicates an update value of $+1$ and black indicates an update value of -1. For CIFAR-10, the three separate colour channels have been combined into a colour image

MNIST-DNN				
λ	0.001	0.01	0.1	0.2
control	98.77	98.59	94.61	83.31
updates	98.79	98.56	94.68	83.14
MNIST-DNN-AUG				
λ	0.001	0.01	0.1	0.2
control	99.03	98.78	95.10	84.24
updates	98.91	98.78	95.06	84.34
MNIST-CNN				
λ	0.001	0.01	0.1	0.2
control	99.47	99.01	84.16	61.10
updates	99.45	99.21	89.01	66.93
MNIST-CNN-AUG				
λ	0.001	0.01	0.1	0.2
control	99.54	99.33	91.47	71.55
updates	99.62	99.39	92.20	73.48
CIFAR-10				
λ	0.001	0.01	0.1	0.2
control	89.82	82.40	51.20	40.53
updates	90.18	87.81	66.86	52.57
CIFAR-10-AUG				
λ	0.001	0.01	0.1	0.2
control	92.98	91.49	74.24	59.50
updates	93.00	91.33	74.06	58.73
CIFAR-100				
λ	0.001	0.01	0.1	0.2
control	64.61	53.26	26.19	19.01
updates	65.55	62.59	40.92	27.38
CIFAR-100-AUG				
λ	0.001	0.01	0.1	0.2
control	69.78	67.76	48.47	32.77
updates	70.12	68.08	48.29	32.12

Table 5: Test accuracy on the adversarial versions of the benchmark test sets. DNN indicates a fully-connected Deep Neural Network, CNN indicates a Convolutional Neural Network, whilst AUG indicates that dataset augmentation was used.

is an effective manner of increasing the robustness of a model to the effects of adversarial examples.

We conducted experiments on benchmark computer vision dataset to show empirically, with statistical significance, that our method improves the robustness against adversarial examples when compared to the Fast Gradient Sign method, whilst not negatively affecting the normal learning performance.

Work has been done on optimization methods in Stochastic Gradient Descent, with the development of novel *update rules* that improve on the simple update method based on a constant learning rate rate [21, 16, 1, 8, 18, 10]. It is therefore worth investigating whether these update rules may be applied to the residual gradient in such a manner as to improve either the resilience to adversarial examples, improve the positive effect on the learning of generalisations, or both. Another interesting element of future work would be to explore

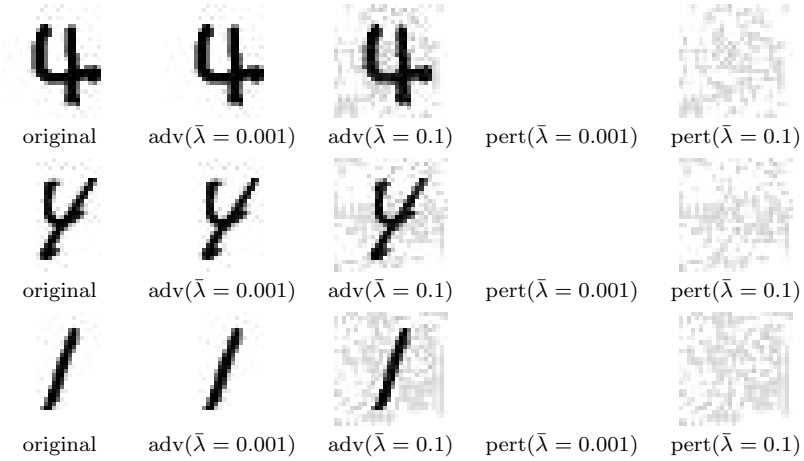

Fig. 3: Examples of adversarial images for the MNIST dataset at different values of $\bar{\lambda}$ (adv = adversarial image with perturbations, pert = perturbations alone, without the original image)

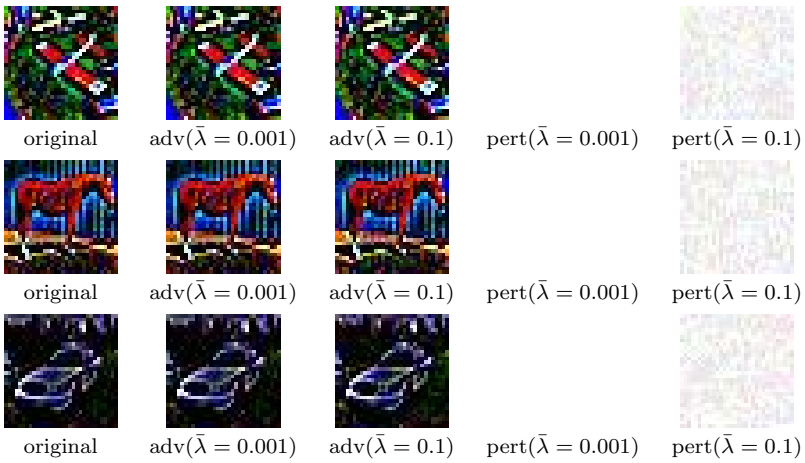

Fig. 4: Examples of adversarial images for the CIFAR-10 dataset at different values of $\bar{\lambda}$ (adv = adversarial image with perturbations, pert = perturbations alone, without the original image). The "original" image is after ZCA-whitening and preprocessing.

the effects of adding the error with respect to the input to the error function as a regularisation term.

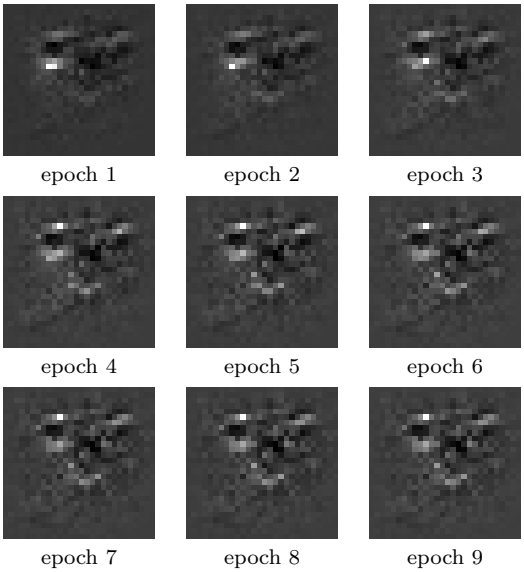

Fig. 5: Normalized updates applied to the input at various epochs on MNIST

8 Compliance with Ethical Standards

Funding: The equipment for these experiments was funded by a grant from NVidia Corporation.

Conflict of Interest: George D Magoulas has received research grants from Nvidia Corporation. Alan Mosca owns stock in Alphabet, Facebook, NVidia and Twitter.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Anastasiadis, A.D., Magoulas, G.D., Vrahatis, M.N.: An efficient improvement of the rprop algorithm. In: Proceedings of the First International Workshop on Artificial Neural Networks in Pattern Recognition (IAPR 2003), University of Florence, Italy, p. 197 (2003)
2. Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep, big, simple neural nets for handwritten digit recognition. Neural computation **22**(12), 3207–3220 (2010)
3. Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., Lempitsky, V.: Domain-adversarial training of neural networks. Journal of Machine Learning Research **17**(59), 1–35 (2016)
4. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
5. Hahnloser, R.H., Sarpeshkar, R., Mahowald, M.A., Douglas, R.J., Seung, H.S.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature **405**(6789), 947 (2000)

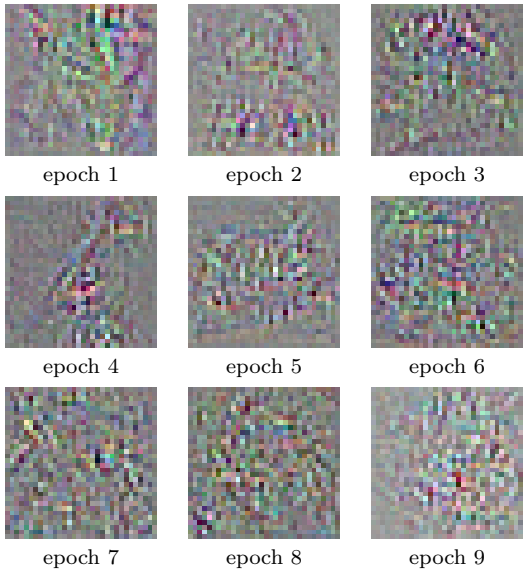

Fig. 6: Normalized updates applied to the input at various epochs on CIFAR-10

6. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural Networks, 1989. IJCNN., International Joint Conference on, pp. 593–605. IEEE (1989)
7. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
8. Igel, C., Hüsken, M.: Improving the Rprop learning algorithm. In: Proceedings of the second international ICSC symposium on neural computation (NC 2000), vol. 2000, pp. 115–121. Citeseer (2000)
9. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
10. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
11. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
12. LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**, 310 (1995)
13. Lecun, Y., Cortes, C.: The MNIST database of handwritten digits URL http://yann.lecun.com/exdb/mnist/
14. Miikkulainen, R., Dyer, M.G.: Natural language processing with modular pdp networks and distributed lexicon. Cognitive Science **15**(3), 343–399 (1991)
15. Mosca, A., Magoulas, G.: Deep incremental boosting. In: C. Benzmuller, G. Sutcliffe, R. Rojas (eds.) GCAI 2016. 2nd Global Conference on Artificial Intelligence, *EPiC Series in Computing*, vol. 41, pp. 293–302. EasyChair (2016)
16. Mosca, A., Magoulas, G.D.: Adapting resilient propagation for deep learning. UK Workshop on Computational Intelligence (2015)
17. Mosca, A., Magoulas, G.D.: Learning input features representations in deep learning. In: Advances in Computational Intelligence Systems, pp. 433–445. Springer (2017)
18. Mosca, A., Magoulas, G.D.: Training convolutional networks with weight-wise adaptive learning rates. In: ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ. (2017)

19. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against deep learning systems using adversarial examples. arXiv preprint arXiv:1602.02697 (2016)
20. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: Security and Privacy (SP), 2016 IEEE Symposium on, pp. 582–597. IEEE (2016)
21. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: proceeding of the IEEE International Conference on Neural Networks, pp. 586–591. IEEE (1993)
22. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cognitive modeling **5**(3), 1 (1988)
23. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis (2003). URL http://research.microsoft.com/apps/pubs/default.aspx?id=68920
24. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
25. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)